

# CellSpeak

## User Manual

May 2017

*Cells run in parallel,  
On all cores,  
On interconnected systems,  
As one program*

## Table of Contents

1	Introduction .....	3
2	Where is the software.....	4
3	Installing CellSpeak .....	5
4	Compiling a CellSpeak application .....	6
4.1	Introduction .....	6
4.2	The project file .....	6
4.3	The compile command.....	8
4.4	Compiling a single source file.....	9
5	The Compiler Output File .....	10
5.1	Load and Compile sections .....	10
5.2	After a successful compilation .....	13
5.3	Bytecode output .....	14
6	Running a CellSpeak application .....	18
6.1	Introduction .....	18
6.2	Starting up a CellSpeak application.....	18
6.3	Files and dependencies .....	19
7	Using the Notepad++ plugin .....	20
7.1	Introduction .....	20
7.2	Installation .....	20
7.3	Command Overview.....	23
7.4	Compiling and running a CellSpeak application.....	23
7.5	Settings.....	24
7.6	Debugging .....	25
Appendix 1	Compiler messages overview.....	27

# 1 Introduction

This manual gives explains how to compile and run a CellSpeak program.

There are four different applications explained in this manual:

1. CellSpeak: the command line compiler
2. CellRun: the virtual machine to run a CellSpeak program
3. CellSpeakPlugin: the CellSpeak plugin for Notepad++

**CellSpeak** is a command line compiler. Its input are source files and other packages, and it produces a CellSpeak bytecode file as its output when the compilation was successful. It has a few switches to set options.

**CellRun** is the virtual machine needed to run a compiled CellSpeak program.

**CellSpeakPlugin** is a piece of software that needs to be integrated into Notepad++. It allows to compile files, to find and correct errors, to run the compiled application and to debug the application all from within Notepad++. CellSpeak plugin is only the first of several plugins that will become available to integrate CellSpeak in other IDE's.

There is also another application that is part of the distribution: **TestBench**. It is a platform for developing and testing new features of the language and/or Virtual Machine. If you are not a developer of the Compiler or the Virtual Machine, you do not need this tool. The use of TestBench is explained in the *CellSpeak Design Manual*.

## 2 Where is the software

There are two downloads available: the first one contains the compiled tools for working with CellSpeak and the second one contains also the source code for these tools.

You only need to download the source code if you want to contribute to the software – or if you are curious about how it is designed. The CellSpeak source can be downloaded from GitHub.

If you only want to use CellSpeak as a development tool, you do not need access to the source code of CellSpeak. In that case you can download the **CellSpeak Distribution**. It is a distribution with the tools and examples necessary to start working in CellSpeak. This distribution is available directly from the CellSpeak website but also on GitHub.

### 3 Installing CellSpeak

The installation of the CellSpeak distribution is straightforward, it uses an installer and will install the software, examples and tools on your computer.

The description of the installation can be found in the readme file that accompanies the distribution.

## 4 Compiling a CellSpeak application

### 4.1 Introduction

To compile a CellSpeak application you need a project file:

```
cellspeak -p projectfile
```

The project file contains the sources and packages that are needed for a compilation. Normally a project will have a dedicated project file, but for single source file applications it is possible to use the same default project file.

Much of the examples included in the distribution consist of a single source file and the same list of packages, so most of the examples use the same default project file.

Let's first have a look at the content of a project file

### 4.2 The project file

The basis for each CellSpeak project is the CellSpeak project file. The project file has the extension *.celprj* and contains a number of arrays of strings:

```
-- Project file for the rotating cubes
group Project

-- The source files for the packages used
const byte[] Source[] = [
    "sourcefile_1.celsrc",
    ..
    "sourcefile_n.celsrc",
]

-- The package files for this project - also load the symbols
const byte[] FullPackage[] = [
    "fullpackage_1.celbyc",
    ..
    "fullpackage_m.celbyc"
]

-- packages to load without symbols
const byte[] Package[] = [
    "package_1.celbyc",
    ..
    "package_p.celbyc",
]

-- list of aliases for external libraries
const XLib is record
    libalias_1 = "externallib_1.dll"
    ..
    libalias_q = "externallib_q.dll"
end
```

The contents of a project file are always part of a group called project.

The first array of strings is called **Source** and it is the list of source files that are part of the project.

The second array is called **FullPackage** and it is the list of compiled packages that this project needs for compilation. A package contains one or more compiled modules. A full package is a package for which the compiled code and the symbol information for the package – the types, functions etc. – will be loaded by the compiler.

The third array is called **Package**. This array contains the list of packages that will be loaded without the symbol information. If a package is loaded without the symbol information, you still have access to the exported designs of that package, i.e. you can create cells based on these design and the compiler can check if the message handlers match the messages being sent to the designs of the package.

The package lists have to be complete in the project file. If your project needs a package A that needs a package B, then both package A and package B have to be listed. Secondary dependencies should always be listed in the **Package** list and not in the **FullPackage** list, because the source of the project does not depend on symbols in the secondary package.

It could be that package A depended on the symbols of package B, i.e. A has a hard dependency on B, but that fact is then already saved in package A, so the compiler will be aware of it and will check the versions of the packages.

**Important** : it is possible to compile a project without listing the packages with which it has a soft dependency – the packages with which there is a hard dependency always have to be listed – the compiler will then not be able to do a message check, and will not be able to resolve all designs that are instantiated in the source. The resolution of the designs will in any case be done when the package is loaded by the Virtual Machine.

The last structure in the project file is a record, **XLib**. In this record you can define fields as aliases for external library files. If you then have a source file where you import the library file, you can use the alias. If for example the following field is defined in the XLib record:

```
const XLib is record
  Direct3D11Lib = "..\\TestBench(winx86)\\Debug\\Direct3D11(winx86).dll"
end
```

then you can refer to that file in a group declaration (and only there) as follows:

```
group D3D11 is lib Direct3D11Lib
```

In this way file names for a project can all be grouped in the project file.

Not all structures have to be present in a project file, and the order of the structures is not important.

The following is an example of a project file:

```
group Project

-- The source files for the package
const byte[] Source[] = [
```

```

    "Math.celsrc",
    "Strings.celsrc",
    "Structures.celsrc",
    "Windows.celsrc",
    "System.celsrc",
    "Timers.celsrc",
    "ConxMgr.celsrc",
    "FileHandling.celsrc"
]

-- The package files for this project - also load the symbols
const byte[] FullPackage[] = [
    "..\\Packages\\Platform\\Platform.celbyc",
    "..\\Packages\\D3D11Graphics\\D3D11Graphics.celbyc"
]

-- Also needed - but without symbols
const byte[] Package[] = [
    "..\\Packages\\Editor\\Editor.celbyc"
]

const XLib is record
    WindowsLib = "..\\testbench(winx86)\\debug\\Windows(winx86).dll"
    SystemLib = "..\\testbench(winx86)\\debug\\VirtualMachineLib(winx86).dll"
end

```

In the next paragraphs we have a look at the command to compile a project.

### 4.3 The compile command

The following command is used to compile a CellSpeak project:

```
cellspeak -p <project file> [ -b <byte code> -o <output file> -c -m -x -?]
```

The name of the compiler is cellspeak and the options have the following meaning:

option	explanation
-p	<project file> if no extension .celprj is assumed
-b	<byte code> file, if not present the project filename with extension .celbyc is used.
-j	save the result in JSON format – same name as byte code file, extension .celjson
-o	<output file> (errors and info), if not present stdout is used.
-a	Same as -o, but the file will be appended.
-c	output bytecode in a readable format - goes to the output file
-m	output bytecode in a readable format mixed with source code
-x	output content of the external libraries
-f	do a static message flow check
-?	prints this usage overview

In the -p option the project file is specified, if no extension is given then the default extension *celprj* is assumed.

The -b option allows to specify a different byte code file then the default one used by the compiler. The default proposed is *projectname.celbyc*.



The `-j` option instructs the compiler to generate a JSON textfile that contains the same code, but in JSON format.

The output generated by the compiler can be redirected to a file with the `-o` option, if not present output is directed to `stdout`. In the next section we will have a detailed look at what output the compiler produces.

The `-a` option instructs to append the output to the file given, instead of overwriting it.

The `-c` option instructs the compiler to print a readable version of the bytecode produced.

The `-m` option instructs the compiler to print the source code interspersed with the readable byte code generated. This can be helpful when debugging the code.

The `-x` option will print the content of the external libraries used in the compilation. The compiler will print all classes and methods it has found in the library.

The `-f` option is to instruct the compiler to do a message check, namely to check whether each message sent in the source has a message handler somewhere in the packages. This allows to detect message for which it cannot find any message handler, and message handlers for which - in this source code - never a message is generated.

These conditions are not necessarily errors, but it can help to spot obvious (typing) mistakes.

#### 4.4 Compiling a single source file

If the project to compile only has a single source file, then the compiler can also use a default project file, in the directory where the source file is. The command to compile does not contain the project file, but the single source file that needs to be compiled.

```
cellspeak <source file> [ -b <byte code> -o <output file> -c -m -x -?]
```

In this way it is not necessary to have a dedicated project file for each small project.

## 5 The Compiler Output File

In this section we have a closer look at the output generated by the compiler.

If the compilation is successful, the compiler will generate two files: the file with the compiled bytecode and a text file with the results of the compilation. If the option `-j` was specified, the compiler generates a third file, the compiled bytecode in JSON format.

If the compilation is not successful the compiler will only generate the output text file, indicating the errors that were found in the project.

In the following paragraphs we look at the content of this text output file. The file consists of several sections. Each section is preceded by a title like in this example for the *external libraries*:

```
-- External Libraries-----
```

Let us now have a look at the different sections of the file. The first sections are the load and compile sections, and contain output generated when the packages are loaded and when the source files are being compiled.

### 5.1 Load and Compile sections

Below you find an example of the first sections of an output file:

```
-- Compiler Output -----
-- Mon May 22 11:08:57 2017

-- Load packages -----
Loading Package and Symbols from \Platform\Platform.cellbyc
  Parse xlib c:\debug\Windows(winx86).dll for symbols ... success
  Parse xlib c:\debug\VirtualMachineLib(winx86).dll for symbols ... success
Loading Package from C:\Development\CellSpeak\Packages\Editor\Editor.cellbyc
  Parse xlib c:\Windows\syswow64\Scintilla.dll for symbols ... success
  Parse xlib c:\debug\EditorLib(winx86).dll for symbols ... success

-- Global module list -----
1.c:\mydir\platform.cellprj Version: 00.00.00 TimeStamp: Wed Mar 22 08:37:44 2017
2.c:\mydir\editor.cellprj Version: 00.00.00 TimeStamp: Wed Mar 22 08:37:44 2017
```

The first section simply has a timestamp for when the compilation took place.

The second section lists the packages that are loaded for the compilation – the packages listed in the project file. If the package references an external library, then that library is also parsed to extract the classes and methods.

When all packages are loaded, the compiler lists all the modules that it will use in the compilation together with their version number and timestamp.

The next section is the Compiling section. For each file the compiler will either output *success* or a list of errors that were encountered during the compilation.

```
-- Compiling -----
Source file C:\Development\CellSpeak\Examples\02 CellSpeak Types.cellsrc
```

## Success

Below is an example of the output of a compilation of several files, that also contain templates.

```
-- Compiling -----
Source file C:\Platform\Math.cellsrc
Source file C:\Platform\Strings.cellsrc
Compiling template StringComparison at line 245
Compiling C:\Platform\Strings.cellsrc - continue from line 245
Source file C:\Development\CellSpeak\Packages\Platform\Structures.cellsrc
Source file C:\Development\CellSpeak\Packages\Platform\Windows.cellsrc
    Parse xlib c: debug\Windows(winx86).dll for symbols ... success
-- Info  EXT 013 at line 17.Multiple equivalence links.Type unsigned int is
    already linked to type word32 - (new link to cell)
Compiling template HashTable at line 75
Compiling C:\Platform\System.cellsrc - continue from line 75
Source file C:\Development\CellSpeak\Packages\Platform\Timers.cellsrc
-- Info  GRP 001 at line 9.This group exists already (if not intended: choose a
    different name).Name: System
-- Info  CST 011 at line 23.Cast to a derived record type - allowed, but
    potentially dangerous. From Record to TimerRequest
Source file C:\Development\CellSpeak\Packages\Platform\FileHandling.cellsrc
Success.
```

The compiler signals when the compilation process switches from a source file to a template and back. As we will see the compiler can output errors, but it can also output *info* if it encounters something that is not an error, but worth signaling. The format of an info message is

```
-- Info  ABC XXX at line <n>. <text message>
```

The code *ABC XXX* is composed of two parts: the category code, *ABC*, and the number of the message in that category. It makes identifying and locating errors somewhat easier. The list of errors is given in the appendix to this document.

It is very well possible to see files appear several times in the list of files being compiled. This is simply because at the start the compiler does not know the compile order and the interdependency between the files. The compiler will start a compilation and if it sees that the file depends on other groups that are not yet compiled, it will abandon the compilation and start with another file. The compiler will work out the dependencies in this way and finally compile the files in the right order.

Let's have a look now at the output of the compiler when errors were encountered:

```
-- Compiling -----
Compiling C:\D3D11Graphics\Direct3D11.cellsrc
Compiling C:\D3D11Graphics\Camera.cellsrc
Compiling C:\D3D11Graphics\Light.cellsrc
Compiling C:\D3D11Graphics\ColorAndMaterial.cellsrc
Compiling C:\D3D11Graphics\Mesh.cellsrc
Compiling C:\D3D11Graphics\Shape.cellsrc
Compiling C:\D3D11Graphics\Direct3D11.cellsrc
    Parse xlib C:\Debug\Direct3D11(winx86).dll for symbols ... success
-!- Syntax Error at line 12.
    'nothing' cannot be handled here [token 'UNKNOWN_NAME' in parser state 641].
See the syntax rule(s) below.
    Rule 586 type_equivalence : library_type_name IS * TYPE_VOID
```

```

Rule 587 type_equivalence : library_type_name IS * type_spec
Compiling C:\D3D11Graphics\Camera.cellsrc
Compiling C:\D3D11Graphics\Light.cellsrc
Compiling C:\D3D11Graphics\Mesh.cellsrc
-!- Syntax Error at line 25.
    'MeshType' cannot be handled here [token 'UNKNOWN_NAME' in parser state 659].
See the syntax rule(s) below.
    Rule 583 type_definition : TYPE PRIORITY_NAME IS * type_spec with
    Rule 584 type_definition : TYPE PRIORITY_NAME IS * type_spec
Compiling C:\D3D11Graphics\Shape.cellsrc
-!- Syntax Error at line 58.
    'MeshClass' cannot be handled here [token 'UNKNOWN_NAME' in parser state 595].
See the syntax rule(s) below.
    Rule 111 design_header : design_name OPBR * CLBR
    Rule 113 design_header : design_name OPBR * formal_design_parameter_list CLBR
Compiling C:\D3D11Graphics\Camera.cellsrc
-!- Error DCL 002 at line 36.Unknown token: CameraClass

```

In the results above, there are several syntax errors that are signaled by the compiler. Error messages always start with `-!-`.

Broadly speaking there are two groups of errors: syntax errors and semantic errors.

A syntax error is an error against the grammar rules of the language. When it encounters a syntax error, the compiler lists the rule(s) that could apply up to the error, and indicates up to which point the rule(s) had been parsed with a star.

In the first syntax error for example, the compiler had two rules that it was working with, rule 586 and rule 587 in this case, and expected either a terminal `TYPE_VOID` (text image `void`) or a `type_spec`. Instead it received a word *nothing* that the parser did not recognize `UNKNOWN_NAME`.

The rule numbers and the parser state refer to the rules and states as can be found in the `CellSpeakGrammr.txt` file, but normally the syntax error messages are clear enough and there is no need to consult the grammar file.

There is also another error at the end `DCL 002` for a name that was not recognized in the source file.

It is also possible that during compilation, the compiler did not find all the groups that are in the `use-list` at the start of the source files. In that case, the compiler will output a list of missing groups:

```

-- Missing groups -----
group Shapes in C:\Development\CellSpeak\Examples\10 Solar System.cellsrc
group Rotations in C:\Development\CellSpeak\Examples\10 Solar System.cellsrc

```

Normally this means that either a required package has not been included, or that the group name was mis-spelled.

If the compiled files contain instantiations of templates, i.e. templates where the dummy types have been replaced by the real types, then the expanded template is also printed. If there were any errors in these templates, this will help to locate the error.

If the compilation was not successful, then this is where the output file ends. When the errors are corrected, the compilation can be launched again.

## 5.2 After a successful compilation

After a successful compilation, the compiler will load external library files. If the files were already loaded, this is mentioned also. The XLibHandle is for information only.

```
-- External Libraries-----  
Library <Windows(winx86).dll> is preloaded - XLibHandle 1  
Library <VirtualMachineLib(winx86).dll> is preloaded - XLibHandle 2  
Library <Direct3D11(winx86).dll> is preloaded - XLibHandle 3
```

Next, if the `-f` flag was used, the compiler will do a *static message flow check*. This means that it will check that for each message sent, there is a corresponding handler. It will also list if a handler is never used. The check is static because it purely based on the source code – the compiler cannot check if the message is sent at the right cell, but it will allow to find typing errors or parameter errors in the messages.

The output from the message check looks as follows:

```
-- Messages with missing handlers-----  
1. stdout(cell) in handler stdout.Get (module name)  
2. stderr(cell) in handler stderr.Get (module name)  
3. Service.InvalidProvider(byte array) in handler Service.Offer (module name)  
4. Service.Exists(byte array,cell) in handler Service.Offer (module name)  
5. Service.Added(byte array,cell) in handler Service.Offer (module name)  
6. Service.NotAdded(byte array,cell) in handler Service.Offer (module name)  
7. Memory.Usage(record [ int32,int32, ]) in handler Memory.GetStatus (module name)  
8. Close(cell) in destructor for RemoteHandler (module name)  
  
-- Unique Fixed Messages Sent-----  
1. Service.Revoke(byte array,cell)  
   Handled by System.SystemDesign()  
2. TCPIP.Listen(byte array)  
   Handled by System.ConxManagerDesign(cell)  
3. Memory.Usage(record [ int32,int32, ])  
   NO HANDLER FOUND  
4. Service.Added(byte array,cell)  
   NO HANDLER FOUND  
5. Subscribe.Interval(word32,byte array)  
   Handled by System.TimerDesign()  
6. Service.Offer(byte array,cell)  
   Handled by System.SystemDesign()
```

The first list is a list of messages for which no handler has been found. It gives the name of the message, the parameters and where the message was sent from, e.g. function h (module x).

Note that if the package that contains the handler for this message was not included in the package list, this does not invalidate the compilation, but the message will be listed here.

The second list is a list of all unique messages, and of the handlers where these messages are handled. A message can of course have more than one handler.

Next the compiler will show some compilation timings in ms. As the CellSpeak compiler is a single pass compiler, compilation is normally quite fast.

```
-- Compiler Timings -----  
CellSpeakClass::CompileStream was called 21 times total duration 56 ms  
ProjectClass::LoadAllPackages was called 1 times total duration 0 ms  
ProjectClass::CompileEachFile was called 1 times total duration 65 ms  
ProjectClass::AfterCompilation was called 1 times total duration 0 ms
```

The routine *CompileEachFile* is where the compilation takes place – so total compilation time was 65 ms. It called *CompileStream* 21 times for a total time of 56 ms. As we have seen *CompileStream* can be called more than there are source files in the project, because the compiler has to work out the compilation order.

After compiling and checking the messages, there is a link step.

The following line just confirms that the compiler has found all the designs that are used in the newly compiled bytecode. If not all designs have been resolved, the compilation will still be successful, but when loading the package for execution, the missing designs will have to be part of some of the other packages already loaded by the Virtual Machine.

```
-- Unresolved Designs -----  
There are no unresolved designs in module c:\editor\editor.cellprj  
There are no unresolved designs in module 02 CellSpeak Types
```

If there are unresolved designs they are listed:

```
-- Unresolved Designs -----  
Editor.MenuWindow(T) Local Unit 175
```

The next section is the list of exported designs of the compiled module:

```
-- Exported Designs -----  
Design System.AvatarList() Unit 185  
Design System.ConnManagerDesign(C) Unit 133  
Design System.HTTPList() Unit 186  
Design System.RemoteHandler(PCPC) Unit 147  
Design System.SystemDesign() Unit 126  
Design System.TimerDesign() Unit 134  
Design System.TimerTest() Unit 173
```

These are the designs that can be instantiated by other packages that use this package. The unit numbers are for information only. The parameters are given as a parameter signature, where each character represents a generic type.

Next, if requested by using **-b** or **-m** option, the compiler will output the bytecode for the source files it has compiled.

### 5.3 Bytecode output

The bytecode output starts with the name of the modules and the global module number.

After that the bytecode follows for each code unit in the module. A code unit can be a function, a handler, a design, a constructor etc. The header for the unit also gives the address where the bytecode starts and the length of the unit.

After that, all the bytecode instructions in the unit are listed. The following is an example of the bytecode output:

```
-----  
-- Module 10 Solar System (4)  
-----  
<UNIT 1> DESIGN Sphere(fP) bc: 0x0896AFD0 (900 bytes)  
-----  
<0000> CEL_DESC      unit 1 ip 0492 ip 0516 ip 0000 ip 0000 ip 0000 int 188  
<0032> PTR_MOVE      st 0140 st 0068  
<0044> CEL_INHERIT  SharedMesh unit 2 ptr 0x088D61C0 st 0076  
<0080> FLO_MOVE      st 0148 st 0064  
<0092> I32_SET       st 0152 i32 0  
<0104> I32_SET       st 0156 i32 0  
<0116> FLO_MOVE      st 0160 pr 0000  
<0128> I32_SET       st 0164 i32 0  
<0140> FLO_MOVE      st 0168 st 0064  
<0152> I32_SET       st 0172 i32 0  
<0164> FLO_MOVE      st 0176 pr 0004  
<0176> I32_SET       st 0180 i32 0  
<0188> I32_SET       st 0184 i32 0  
<0200> FLO_MOVE      st 0188 st 0064  
<0212> FLO_MOVE      st 0192 pr 0008  
<0224> I32_SET       st 0196 i32 0  
<0236> I32_SET       st 0200 i32 0  
<0248> I32_SET       st 0204 i32 0  
<0260> I32_SET       st 0208 i32 1  
<0272> FLO_MOVE      st 0212 st 0148  
<0284> I32_TO_FLO    st 0216 st 0152  
<0296> I32_TO_FLO    st 0220 st 0156  
<0308> FLO_MOVE      st 0224 st 0160  
<0320> I32_TO_FLO    st 0228 st 0164  
<0332> FLO_MOVE      st 0232 st 0168  
<0344> I32_TO_FLO    st 0236 st 0172  
<0356> FLO_MOVE      st 0240 st 0176  
<0368> I32_TO_FLO    st 0244 st 0180  
<0380> I32_TO_FLO    st 0248 st 0184  
<0392> FLO_MOVE      st 0252 st 0188  
<0404> FLO_MOVE      st 0256 st 0192  
<0416> I32_TO_FLO    st 0260 st 0196  
<0428> I32_TO_FLO    st 0264 st 0200  
<0440> I32_TO_FLO    st 0268 st 0204  
<0452> I32_TO_FLO    st 0272 st 0208  
<0464> M4F_MOVE      pr 0012 st 0212  
<0476> FCV_INIT      pr 0092 0x80000000 st 0000  
<0492> CEL_END       Sphere(fP)  
<0516> MSG_TABLE      int 8 ip 0788 size 256  
    0016 Draw(P)          unit 7 bytecode 0x088D6560 hash 0xd0db1e3  
    0023 SetMaterial(z)   unit 8 bytecode 0x088D6720 hash 0xd8dae5a  
    0037 Move(x)          unit 9 bytecode 0x088D6680 hash 0xdaf4074  
    0044 NewFrame(w)      unit 3 bytecode 0x089AB010 hash 0x19e146d1  
    0055 MoveTo(x)        unit 10 bytecode 0x088D6630 hash 0x3680a317  
    0064 SetOrbit(xf)     unit 4 bytecode 0x089AB050 hash 0x5a55c20f  
    0076 SetWorldMatrix(j) unit 11 bytecode 0x088D6520 hash 0xa383b258  
    0093 SetMaterial(Rzzz#) unit 12 bytecode 0x088D66E0 hash 0xecd107fd
```

```

<0788> FLW_STRTBL  size 96 int 16 int 0
<0804>             0x77617244 0x53005000 0x614d7465 0x69726574
<0820>             0x7a006c61 0x766f4d00 0x00780065 0x4677654e
<0836>             0x656d6172 0x4d007700 0x5465766f 0x0078006f
<0852>             0x4f746553 0x74696272 0x00667800 0x57746553
<0868>             0x646c726f 0x7274614d 0x6a007869 0x74655300
<0884>             0x6574614d 0x6c616972 0x7a7a5200 0x00237a7a

```

```

-----
<UNIT 2> DESIGN Shape.SharedMesh(P) redirected to (2.37)
-----

```

The bytecode instructions themselves are explained in the *CellSpeak Design Manual*. As an example we take the following instruction:

```

<0404> FLO_MOVE   st  0256 st  0192

```

This instruction starts at byte offset 404 (decimal) in the unit, and it moves a floating point value from position 192 on the stack to position 256.

Another example is the instruction at position 516, MSG\_TABLE, which is the table with the message handlers for this design.

When doing some detailed debugging, it is often more interesting to have a combination of source code and bytecode:

```

[33]
[34]  -- when a new frame has to be calculated the camera sends NewFrame
[35]  on NewFrame(word Interval) do

```

```

-----
<UNIT 3> MESSAGE NewFrame bc: 0x089AB010 (60 bytes)
-----
<0000> MSG_DESC   unit 3 ip  0040 ip  0000

[36]
[37]      -- ..just execute the frame action for the sphere
[38]      FrameAction( Interval )

<0016> W32_MOVE   st  0116 ms  0000
<0028> FCV_CALL   pr  0092 st  0084

[39]  end

<0040> MSG_END    NewFrame

[40]
[41]  -- Note that the camera also sends the message 'draw', handled by
[42]  -- ancestor design SharedMesh, because processing is the same for all
[43]
[44]  -- A message that sets the particular frame action for the sphere
[45]  on SetOrbit( xyz Axis, float Speed) do

```



```

<UNIT 4> MESSAGE SetOrbit bc: 0x089AB050 (132 bytes)
-----
<0000> MSG_DESC      unit 4 ip  0112 ip  0000

    [46]
    [47]      -- define the body for FrameAction
    [48]      body FrameAction with
    [49]
    [50]      -- Calculate the rotation matrix to use
    [51]      matrix4 RotationMatrix = <matrix4> Rotate.Axis( Axis, Speed)

<0016> V3F_MOVE      st  0180 ms  0000
<0028> FLO_MOVE      st  0200 ms  0012
<0040> FCT_CALL      unit 5 ptr 0x0883D180  st  0148 st  0000
<0064> M3F_TO_M4F    pr  0124 st  0148

    [52]      end is

<0076> FCV_SET      pr  0092 unit 6 ptr 0x089EB050  ptr 0x00000000  pr  0124
-----
<UNIT 6> FUNCTION  FUNCTION 28 bc: 0x089EB050 (56 bytes)
-----
<0000> FCT_DESC      unit 6 ip  0032 ip  0000

    [53]
    [54]      -- Calculate the world matrix
    [55]      WorldMatrix = RotationMatrix * WorldMatrix

<0016> M4F_MULMATR    pr  0012 rc  0000 pr  0012

    [56]      end

<0032> FCT_END      FUNCTION 28

    [57]      end

<0112> MSG_END      SetOrbit

```

The text in blue is the original source code with the line number added, and the text in green is the bytecode as described before.

Normally, inspecting the bytecode output is not required when working with the CellSpeak compiler but it is useful when contributing to the CellSpeak source, or just, if you are curious, to learn more about the CellSpeak bytecode.

## 6 Running a CellSpeak application

### 6.1 Introduction

There are two ways that you can run a CellSpeak application. The first way is the traditional way, by starting a Virtual Machine with the file that you want to run as a parameter. You also will have to add the name of the design you want to instantiate to get the program going.

The second way is by adding a cell to a Virtual Machine that is already running. You then also have to specify that VM in the command. This can be useful if you want to add a new cell to an already running application. That cell can then talk to the other cells to get information, change their behavior etc. But it is also possible to launch a completely unrelated application in that same VM.

### 6.2 Starting up a CellSpeak application

The general format to start a CellSpeak application is as follows

```
cellrun -i -z <process name> -o <output filename> -v -d -t <n>  
        -p <package list> -x <xlib paths> -c <design and parameters> -?
```

option	explanation
-i	forces the virtual machine to run as an interpreter. Normally bytecode is compiled to native code before execution
-z	<process name> does not launch a new VM, but uses the running process
-o / -a	sets the optional output file for a new VM (existing VM keep their current output file): o overwrites, a appends.
-v	sets the verbose flag - more informational messages will be output (to the output file)
-d	-d sets the debug flag (single threaded and interpreted)
-t	n specifies that the virtual machine should use n threads for running the program. Setting the value to 0 or 1 makes it run single threaded. Not specifying it will schedule a default nr of threads.
-p	followed by the list of packages to load. Without extension .celbyc is used."
-x	followed by the paths to look for the XLib files (external library files).
-c	followed by the design to instantiate and the parameters: group.design param1 param2 param3

Some examples:

```
cellrun -p CubeOfCubes -c World
```

In this example the VM will load the bytecode file *CubeOfCubes.celbyc* and create a cell based on the design *World*, which has no parameters.

```
cellrun -z -c PrintGroup.PrinterServer
```

The running VM will search a design called *PrintGroup.PrinterServer* in the bytecodes it has already loaded and if found create a new cell based on that design.

```
cellrun -t 4 -i -v -p Animals -c Hurd.Cows 20 white
```

This command will start of a new VM that will load the file *animals.cellbyc*, search the design *Hurd.Cows* and create a cell based on that design using the an integer (*20*) and a string (*white*), as parameters. The VM will start four threads to run the application and it will run in interpreted mode.

If a design or a file cannot be found *cellrun* will return an appropriate error message.

### 6.3 Files and dependencies

There are three options for the *cellrun* command that require one or more files: the output file, the packages and the xlib list.

The output file is just one file. Using the *-a* option, the file will be appended if it exists or created, using the *-o* option the file will be overwritten or created. The default file extension is *.txt* but you can also set the extension in the command.

The packages in the command contain the modules that are required by the design that is instantiated in the command. We call the module of the design that is being instantiated the *owner module*.

First we look at the situation where the VM is started specifically for this design (i.e. there is no *-z* option).

For each module in the packages will do the following checks: if the module has a hard link with the *owner module*, then the loader will check if the module version and timestamp correspond. If that is the case, the module is loaded, if not, an error message is given. The command will continue to check the other modules, but the design will not be created.

If the module has a soft link with the *owner module*, then the version and timestamp are also checked, but only an informational warning is given if they do not match, and the module will be loaded.

Let us look now at a command with the *-z* option, i.e. where the cell will be created on a running VM.

For modules with hard links, the first check is the same as described before, but now there is also a second check required because the running VM might already have loaded the module. If that is the case, then the version and timestamp of the loaded module are also checked against the data held by the *owner module*, and if there is a mismatch, an error message is given and the new design will not be instantiated.

Also for modules with a soft link the first check is repeated and then there is also a check whether the module has been loaded or not. If that is the case then also the loaded module is checked and, if there is a mismatch, an informational message will be given, but the loaded module will be used.

## 7 Using the Notepad++ plugin

### 7.1 Introduction

Software development is often done using a development environment, or IDE, or using a preferred editor.

As one is writing software, it is always very helpful to get feedback from the compiler as soon as possible and to get hints or help without having to dive into the documentation each time.

Also, when writing software we need to be able to run and debug the software easily from within the same environment.

CellSpeak has a debugging subsystem that allows to set breakpoints, step into code etc.

The CellSpeak compiler can also run in *information mode*, where it will compile a file that is being edited, in order to signal errors to the user as soon as possible or to give information to the user that will improve his productivity, for example about the parameters of a function, the type of a variable etc.

There are many IDE's of course and CellSpeak will be integrated in many over the coming months.

In this chapter we present how CellSpeak has been integrated in Notepad++, a very popular and powerful editor for windows.

### 7.2 Installation

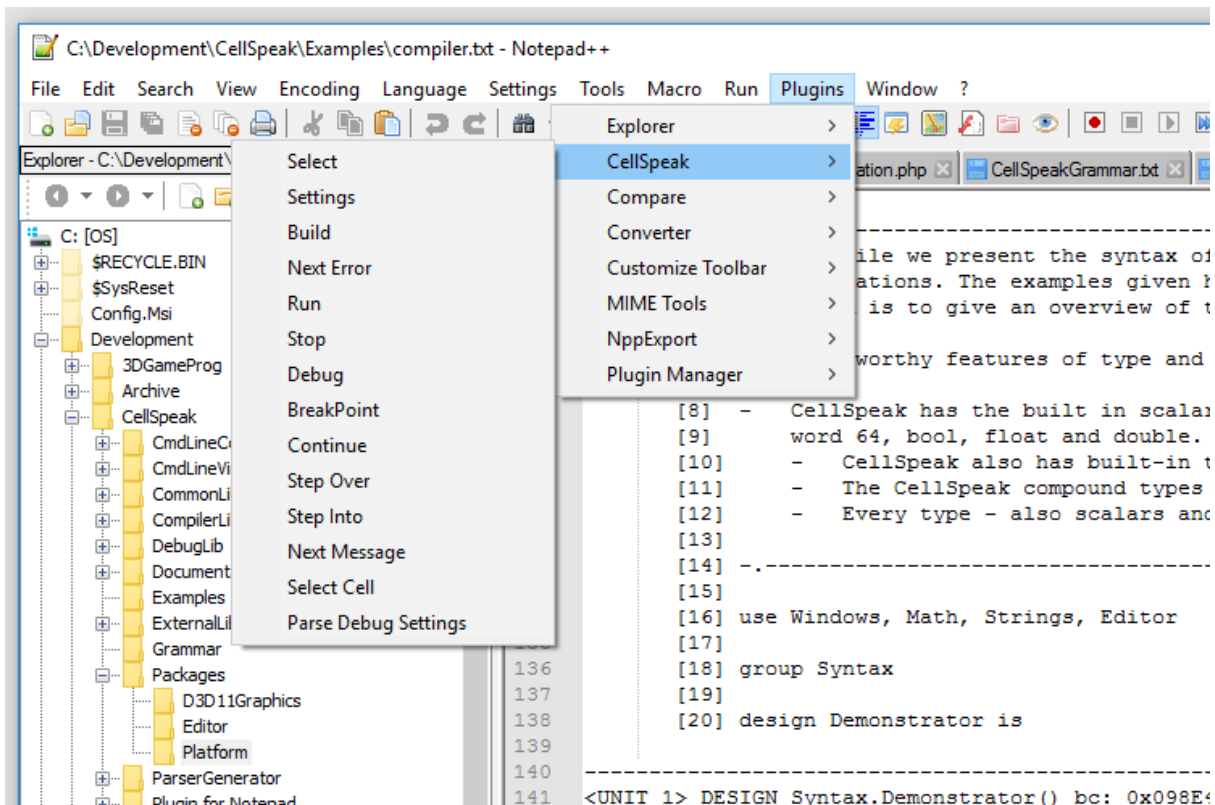
The Notepad++ editor allows to add plugins to supplement the base functionality of the editor, but before you can add plugins, you have to install the editor itself. The installation is straightforward: just go to <https://notepad-plus-plus.org/> and follow the instructions. Most of the plugins are available as 32 bit plugins, so we suggest you download the 32 bit (default) version of Notepad++.

There are many plugins available for Notepad++ that allow you to customize Notepad++. The plugin manager that is a part of Notepad++ can help with finding and installing plugins.

Once you have Notepad++ installed, the installer for CellSpeak will install the additional files needed to use CellSpeak from within Notepad++. The following gives an overview of these files.

All the plugins for Notepad++ are installed in the `C:\Program Files (x86)\Notepad++\plugins` directory. A plugin for Notepad is a DLL file, possibly with a configuration file.

The CellSpeak plugin, called `CellSpeakPlugin.dll` just has to be copied to the plugins directory to be ready for use. The next time Notepad++ is loaded, it will also load the CellSpeak plugin and the commands for CellSpeak will be available from the main menu as shown in the figure below:



## !! LANGUAGE COLORIZER !!

There is however also a plugin for Notepad++, called *CustomizeToolbar*, that allows to add commands to the toolbar for easier access. I found this a useful tool as it avoids the menu drill down when you want to use a command.

From the main page on <https://notepad-plus-plus.org/> you can select the item *resources* and from that page you can select *plugin list* and it will divert you to the page: [http://docs.notepad-plus-plus.org/index.php?title=Plugin\\_Central](http://docs.notepad-plus-plus.org/index.php?title=Plugin_Central). On that page you can find the *Customize Toolbar* plugin and download it.

The *customize toolbar* plugin will be installed in the plugin directory as described above, but the plugin uses the roaming directory of the user to put its configuration. If the user's name is Bill, then the configuration data will be stored in: `C:\Users\Bill\AppData\Roaming\Notepad++\plugins\config`

There are two files that *Customize toolbar* will install there: `CustomizeToolbar.btn` and `CustomizeToolbar.dat`. Additionally, we can install in the same directory the bitmap files that we will use as icons in the toolbar for our commands.

The `CustomizeToolbar.dat` is a binary format used by the plugin, so we do not need to do anything to that file.

The file `CustomizeToolbar.btn` is used to link commands in the menu to an icon in the toolbar. In the case for *CellSpeak*, the content of that file is:

```
Plugins,CellSpeak,Select,,mtb_select_project.bmp
Plugins,CellSpeak,Settings,,mtb_settings.bmp
```

```

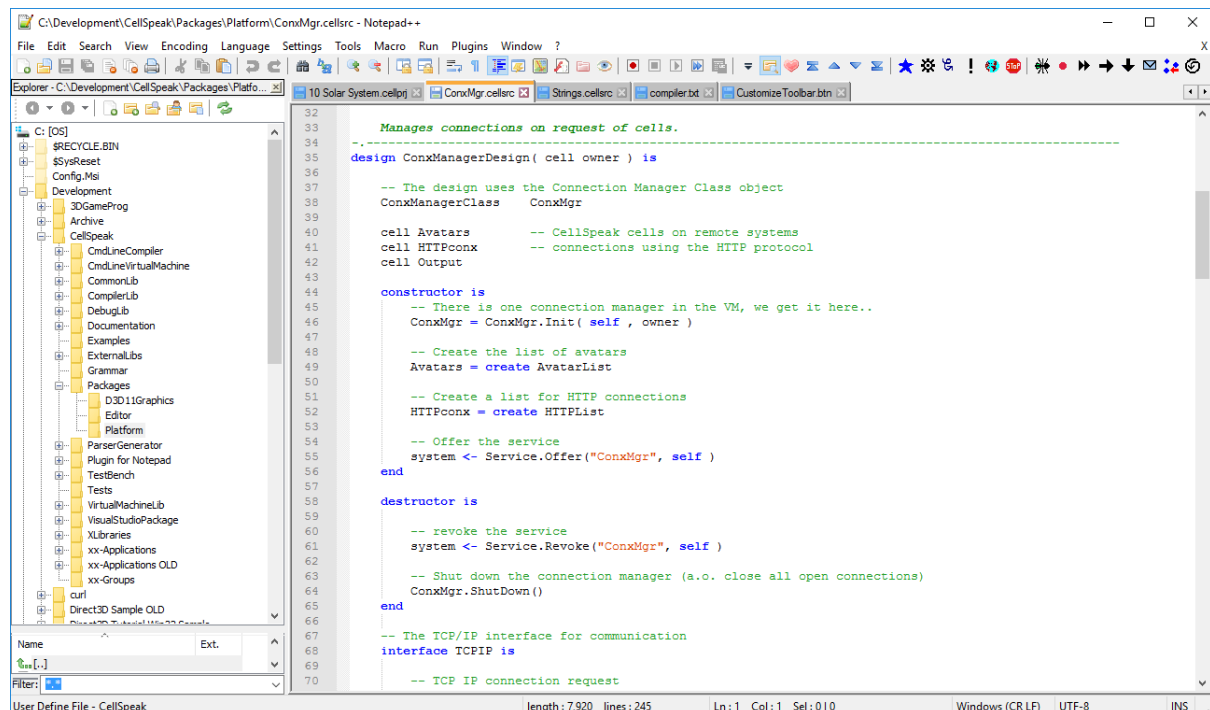
Plugins,CellSpeak,Build,,mtb_compile.bmp
Plugins,CellSpeak,Next Error,,mtb_nexterror.bmp
Plugins,CellSpeak,Run,,mtb_run.bmp
Plugins,CellSpeak,Stop,,mtb_stop.bmp
Plugins,CellSpeak,Debug,,mtb_debug.bmp
Plugins,CellSpeak,BreakPoint,,mtb_brkpt_set.bmp
Plugins,CellSpeak,Continue,,mtb_debug_continue.bmp
Plugins,CellSpeak,Step Over,,mtb_debug_stepover.bmp
Plugins,CellSpeak,Step Into,,mtb_debug_stepinto.bmp
Plugins,CellSpeak,Next Message,,mtb_debug_nextmsg.bmp
Plugins,CellSpeak,Select Cell,mtb_select_cell.bmp
Plugins,CellSpeak,Parse Debug Settings,,mtb_debug_read.bmp

```

The format of each line is: three menu levels, an optional parameter and the bitmap file for the icon. Unused fields are left blank.

If you have already a **CustomizeToolbar.btn** file, you just add the lines for the CellSpeak plugin, otherwise you copy the file from **CellSpeak\Plugin for Notepad\Customize Toolbar** which is part form the distribution. Also all the icon files (.bmp files) can be found in that directory in the distribution. The icon files must be copied to the same directory where the **CustomizeToolbar.btn** is, in our example that would be **C:\Users\Bill\AppData\Roaming\Notepad++\plugins\config**.

When you have installed CustomizeToolbar and copied the files as described above, then the screen for Notepad++ will look as follws, with the added icons to the right:



If we zoom in on the icons:

















where each icon corresponds to a command in the CellSpeak menu.

In the next paragraph, we look at the meaning of each of these commands.

### 7.3 Command Overview

The following table gives an overview and brief description of each of the commands available via the CellSpeak plugin.

command	icon	Compile and Run
<b>Select</b>		Select the project file to use for the compilation, debugging etc. If a source file is selected, the default project file is used if present.
<b>Settings</b>		Displays the settings dialog
<b>Build</b>		The build command. Compiles all the source files in the project.
<b>Next Error</b>		Jumps between the error messages in the output from the compiler and the place in the source where the error occurred.
<b>Run</b>		Runs the application by instantiating the design selected in the settings.
<b>Stop</b>		Stops the Virtual Machine(s)
		<b>Debugging</b>
<b>Debug</b>		Starts a debugging session, same as the run command but the VM runs in interpreted mode in a single thread.
<b>Breakpoint</b>		Toggle Breakpoint
<b>Continue</b>		Continue until next breakpoint or end
<b>Step Over</b>		Step over: execute the next line in the source and stop.
<b>Step Into</b>		Step into: execute the next line in the source. If there is a function call, step into the function call and stop.
<b>Next Message</b>		Next message: execute until the next handler
<b>Select Cell</b>		Select cell: if a cell is selected, the commands are executed for that cell only and not for all cells
<b>Parse Debug Settings</b>		Re-read the breakpoint file

### 7.4 Compiling and running a CellSpeak application

When you build a CellSpeak application, you normally start by creating the project file. If your application is just a single source file, you can also use the default project file in the directory, *default.celprj*.

When you have created the project file, you start writing the source files. Normally that will be an incremental process. When you have something you want to build, you select the *Build* command, and the compiler will compile the sources and generate an output file, *compiler.txt*, where it will list the results in the same format as described for the *cellspeak* command.


If there were errors during the build, the the *Next Error* command allows to jump to the error message (this is a line that starts with `-!-`), the next time you click *Next Error* it will jump to the location of the error in the source file, the next time it will jump to the next error message and so forth.

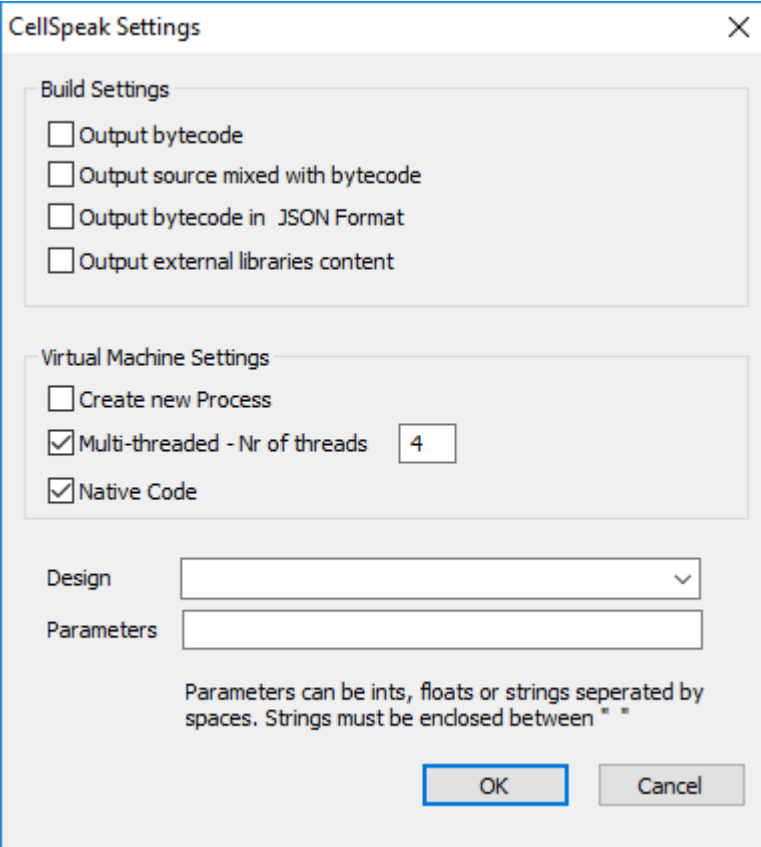
When a project is selected and you are working on a source file that is part of that project, you can hover over a word in the source file and the compiler will give some information about it in a small popup window.

When the build is successful, you can run the application by selecting the *Run* command. As a CellSpeak keeps running normally, waiting for messages, you have to stop the Virtual Machine explicitly by using the *Stop* command.

In the next paragraph we look at the settings that the CellSpeak compiler and the VM use.

## 7.5 Settings

Below you see the dialog box that appears when you select the settings command or 



The image shows a dialog box titled "CellSpeak Settings" with a close button (X) in the top right corner. It is divided into three main sections:

- Build Settings:** Contains four unchecked checkboxes:
  - Output bytecode
  - Output source mixed with bytecode
  - Output bytecode in JSON Format
  - Output external libraries content
- Virtual Machine Settings:** Contains three checkboxes:
  - Create new Process
  - Multi-threaded - Nr of threads (with a text input field containing the number "4")
  - Native Code
- Design and Parameters:** A "Design" dropdown menu is currently empty. Below it is a "Parameters" text input field. A note below the field states: "Parameters can be ints, floats or strings seperated by spaces. Strings must be enclosed between " "".

At the bottom of the dialog are "OK" and "Cancel" buttons.

There are three parts in the dialog box: *Build Settings*, *Virtual Machine Settings* and the selection of a design and its parameters.

The build settings determine the output from the compiler and they correspond to the options for the *cellspeak* command.



The bytecode and the bytecode mixed with the source, will be output to the output file of the compiler, named *compiler.txt*. The JSON format will be output to a files named *projectname.celjson*.

The content of the external libraries will also be output to the file *compiler.txt*

The Virtual Machine settings also correspond with settings for the *cellrun* command.

If you select *Create New Process*, then the next time a design is instantiated, it will be created in its own Virtual Machine. If the option has not been selected, then the design is instantiated in the running Virtual Machine.

Next you can select to run the Virtual Machine in single- or multi-threaded mode, and for multi-thread mode you can select the number of threads to run.

In the last part you can select the design to instantiate from a drop-down list that will be populated after each successful build. The *parameters* edit box allows to specify the parameters for the selected design, if any.

## 7.6 Debugging

CellSpeak has several commands that help in finding bugs in a project.



To run a project in debug mode, you select the *debug* command. This will automatically set the options to run the VM in single-threaded, interpreted mode.

The debug command also uses a debug file, named *projectname.celdbg*, that contains settings for the breakpoints to use. The output of the debugger goes to the file *debug.txt*.

In the file, the breakpoints are stored together with the data that one wants to inspect at that breakpoint. The following is an example of a *celdbg* file:

```
Created on: Wed Mar 22 08:13:14 2017  
  
#break at 38 in "C:\Development\CellSpeak\Examples\10 Solar System.cellsrc"  
Interval, Axis, RotationMatrix
```

The line starting with *#break* states that the execution of the program must stop at line 38 in the file that follows. When the program has stopped it will print three variables: *Interval*, *Axis* and *RotationMatrix*. The *celdbg* file is a text file, so you can add breakpoints and variables to display by editing the file. You can also add breakpoints by using the *BreakPoint* command



The breakpoint command adds a breakpoint to the breakpoint file. The breakpoint will be added for the line in the file where the cursor is at that moment. If you use the breakpoint command on a line that already has a breakpoint, then the breakpoint will be made inactive. Note however that the breakpoint will not be removed from the *celdbg* file, so the next time you load the debug file, it will be activated again.

If you want to remove the breakpoint completely, then you have to delete the breakpoint entry in the *celdbg* file.

Each time a breakpoint is hit, the VM re-reads the *celdbg* file, to see if there were any changes. But sometimes you want to change the breakpoint file, for example because you want to add an extra variable to inspect, while the VM is halted at a breakpoint. In that case you can force a reread of the breakpoint file by using the *Parse Debug Settings* command.



This command forces a re-read of the *celdbg* file. This is useful for when the Virtual Machine is stopped at a breakpoint, and you want to display more data. Variables that were added to the list of variables to inspect, will then also be displayed.



If the VM is stopped at a breakpoint, it is stopped for a particular cell. If the code were the breakpoint was set, is shared by many cells, it is very well possible that the next time the breakpoint fires it is for another cell. This makes it hard to follow what is happening in one particular cell.

The *Select Cell* command allows to tell the Virtual Machine to break only for the same cell. If you select this command, the next time the Virtual Machine breaks, it will be for the same cell for which it was stopped now. When we have selected a cell we call this cell the *target cell*.



The Virtual Machine can be stopped at a breakpoint, but it can also be stopped at every new message handler that is being executed. Selecting the *Next Message* command, will continue the execution until the invocation of the next message handler. In this way one can go from message to message.

The selection of a target cell (see previous command) also has effect on this command. In this way you go from message to message for *one particular cell*.



The *Continue* command, restarts the Virtual Machine and lets it run until the next breakpoint.



The *Step Over* command allows to step through the source line by line without stepping inside functions on the source lines.

The variables that were listed for inspection at the original breakpoint are displayed again after each step.



The *Step Into* command also steps through the source line by line, but also steps inside the functions that in encounters on a line. The variables at the original breakpoint are also displayed at each step taken.

## Appendix 1 Compiler messages overview

The following is an overview of all the error messages the compiler can generate. In most cases the compiler will print some extra information after the error message, such as the variable or type concerned etc.

### // LEXICAL AND SYNTAX ERRORS

```
{"LEX 001", "Input string was not found in the symbol table."},
{"LEX 002", "Character buffer overflow !"},
{"LEX 003", "Invalid character in input stream."},
{"LEX 004", "Badly formed expression in string - will be ignored."},
{"SYN 001", "Token is not acceptable here."},
{"SYN 002", "Is this file a source file ?"},
```

### // GENERAL TYPE ERRORS

```
{"TYP 001", "This type.subtype combination is not allowed."},
{"TYP 002", "Redefinition of an existing type."},
{"TYP 003", "This type cannot have indices."},
{"TYP 004", "The library alias is already in use as a type name."},
{"TYP 005", "The equivalence between the type and the library type exists already."},
{"TYP 006", "The ptr type can only be used to point to records."},
{"TYP 007", "This type is undefined."},
{"TYP 008", "Definition of types in other groups is not allowed."},
{"TYP 009", "This type does not have fields to be renamed."},
{"TYP 010", "The enumeration types are different."},
{"TYP 011", "Indirection is not allowed in type definitions."},
{"TYP 012", "An index selection must be of an integer type"},
{"TYP 013", "An index selection is only valid for an array variable"},
{"TYP 014", "Incorrect index selection for Array"},
{"TYP 015", "A pointer type (.ptr) is not allowed here."},
{"TYP 016", "With + changes are not allowed on the initial record type specification."},
{"TYP 017", "Arrays of type auto are not supported."},
{"TYP 018", "This type does not have a function list."},
{"TYP 019", "'var' function is not possible in the current context."},
{"TYP 020", "Namespace selection has no meaning for a basic type."},
{"TYP 021", "This function exists already for the type."},
{"TYP 022", "This function has not been declared for the type."},
{"TYP 023", "This type is not an enumerated type."},
{"TYP 024", "This enumerator is already in the list."},
{"TYP 025", "Not an enumerator for this type."},
{"TYP 026", "All functions for the built-in types must be defined in the same module."},
{"TYP 027", "Initialisation of data fields in a type definition is not possible."},
```

### // DECLARATION ERROR

```
{"DCL 001", "The variable is already in the symbol table."},
{"DCL 002", "Unknown token: "},
{"DCL 003", "This symbol is already defined."},
```

```
{"DCL 004", "'var' declaration without initialisation."},
{"DCL 005", "This variable is not in scope here."},
{"DCL 006", "Not a function. Use '=' iso 'is' to initialise a data variable."},
```

```
// INTERNAL COMPILER ERRORS
```

```
{"ICE 000", "Default Message."},
{"ICE 001", "Array with multiple unsized indices, ie like A[x,y] and x and y are
unknown."},
{"ICE 002", "Missing conversion record for unary operation."},
{"ICE 003", "Badly formed expression."},
{"ICE 004", "The variable was not found in the symbol table."},
{"ICE 005", "Nesting too deep in symbol definitions (types or functions)."},
{"ICE 006", "Serious problem: the class was found in symbol table, but not the
namespace."},
{"ICE 007", "Fixed size assignment to a constant that has allocation."},
{"ICE 008", "Could not add actual parameter to parameter list for function."},
{"ICE 009", "Could not find nor create namespace."},
{"ICE 010", "Could not add actual parameter to parameter list for function
variable."},
{"ICE 011", "Function calls from string variables are not yet supported."},
{"ICE 012", "Unsupported assignment operator."},
{"ICE 013", "Multiple contexts - function, message - active."},
{"ICE 014", "Exception Handler without any context."},
{"ICE 015", "Could not create exception namespace."},
{"ICE 016", "Expected fixed value or valid data frame for size descriptor."},
{"ICE 017", "User operator functions must have 2 parameters."},
{"ICE 018", "Could not add return type to list"},
{"ICE 019", "Operator code is not a function."},
{"ICE 020", "Unknown instruction while inlining function/operator."},
{"ICE 021", "Could not create class name - while name is not used yet."},
{"ICE 022", "Type mismatch. Should have been caught at the inheritance stage."},
{"ICE 023", "No allocation type specified for this array."},
{"ICE 024", "Variable with square brackets - type should be array."},
{"ICE 025", "Keep variable: local copy exists already."},
{"ICE 026", "Multiple assignments : list has no components/variables"},
{"ICE 027", "Could not allocate code page."},
{"ICE 028", "Assignment of expression lists to unknown variables to be
implemented."},
{"ICE 029", "Could not translate a global module index to a local module
index."},
{"ICE 030", "Badly formed expression, opening bracket missing."},
{"ICE 031", "There should always be a base type for a type index list !"},
{"ICE 032", "Value has no size."},
{"ICE 033", "Inlining failed after positive analysis."},
{"ICE 034", "Could not create unsized array."},
{"ICE 035", "Duplicate field name in record function."},
{"ICE 036", "Missing allocation type."},
{"ICE 037", "Message body - but missing message header."},
{"ICE 038", "Missing constant value for record field."},
{"ICE 039", "Record Arg not a pointer."},
{"ICE 040", "Return argument (pointer) was not moved to stack."},
{"ICE 041", "Missing range in for-loop."},
{"ICE 042", "Array type in 'for each a in A do' missing."},
```

```
{"ICE 043", "'out' mismatch between formal and actual parameter - missing move instruction."},
{"ICE 044", "Capture records in field functions are not allowed."},
{"ICE 045", "Design has already code unit handle."},
{"ICE 046", "Missing namespace for type."},
```

#### // NAMING / SCOPE ETC ERRORS

```
{"GEN 000", "General type of error."},
{"GRP 001", "This group exists already (if not intended: choose a different name)."},
{"GRP 002", "Missing use group."},
```

#### // ARRAY RELATED ERRORS

```
{"ARY 001", "Arrays cannot be assigned - check type of component and array dimension."},
{"ARY 002", "Arrays cannot be assigned because they have a different number of indices."},
{"ARY 003", "The size of the array does not match the size of the inialisation list."},
{"ARY 004", "The array element type and the type of the initialiser in the list are not compatible."},
{"ARY 005", "This type cannot be used as index type."},
{"ARY 006", "This variable is not an array variable. Component Selection is not possible."},
{"ARY 007", "The index should be an integer type or an enumerated type."},
{"ARY 008", "There is a mismatch between the number of indices to select and the number of indices of the array."},
{"ARY 009", "Selection in a constant array is allowed only with constant indices."},
{"ARY 010", "The index value is too big for selection in a constant array."},
{"ARY 011", "Indices for a constant array should be constants."},
{"ARY 012", "This variable is not an array variable and cannot be reallocated."},
{"ARY 013", "This number of indices for this array variable do not match with the declaration."},
{"ARY 014", "There are no valid indices for this array."},
{"ARY 015", "The element type for this array has no size."},
{"ARY 016", "The constant array has no size information."},
{"ARY 017", "For the initialisation of multidimensional arrays the indices must be given."},
{"ARY 018", "Partial declaration of array indices is not allowed - it has to be all or none."},
{"ARY 019", "This arraytype has already a size."},
{"ARY 020", "Missing index list for this type."},
{"ARY 021", "Too many index lists for this type."},
{"ARY 022", "'size' gives the size in bytes of the array without the payload.\n Use elem to get the nr of elements in an array."},
{"ARY 023", "Index ranges have no meaning for allocation."},
{"ARY 024", "There is a mismatch between the number of ranges to select and the dimension of the array."},
{"ARY 025", "The array does not have a compile time size."},
```

#### // FORMAT ERRORS

```
{"FRM 001", "Too many format specifiers in the format string. "},  
{"FRM 002", "Not enough format specifiers in the format string. "},  
{"FRM 003", "No default format specifier for this type. "},  
{"FRM 004", "There can only be one format specifier : [expr , format]."},
```

#### // BUFFER RELATED ERRORS

```
{"BUF 001", "Buffer size must resolve to an integer."},  
{"BUF 002", "This buffer type has already a size."},  
{"BUF 003", "A buffer index list can have only one size."},
```

#### // ASSIGNMENT ERRORS

```
{"ASG 001", "Assignment between these types is not defined."},  
{"ASG 002", "Assignment to this left hand side is not possible."},  
{"ASG 003", "The left hand side is not a reference variable."},  
{"ASG 004", "The right hand side is not a reference nor the constant null."},  
{"ASG 005", "The left hand side and the right hand side are references to  
different types."},  
{"ASG 006", "Cannot create an automatic variable of this type."},  
{"ASG 007", "The number of variables on the left hand side does not match the  
number of results on the right hand side."},  
{"ASG 008", "The two expressions in cond ? expr1 : expr2 are of a different  
type."},  
{"ASG 009", "The right hand side in this reference assignment is not a an  
array."},  
{"ASG 010", "The left hand side in the reference assignment is not a pointer."},  
{"ASG 011", "This variable is not an 'out' parameter."},  
{"ASG 012", "There is no reference variable in left hand side list."},  
{"ASG 013", "The right hand side in the reference assignment is not a pointer."},  
{"ASG 014", "Assignment to 'this' is not allowed."},  
{"ASG 015", "The right hand side is not the constant 'null'"},
```

#### // CELL RELATED ERRORS

```
{"CLS 001", "Re-use of a parameter name in a design definition is not allowed."},  
{"CLS 002", "Definition of a new design outside of its group is not allowed."},  
{"CLS 003", "Redefinition of the design."},  
{"CLS 004", "The code for this design exists already."},  
{"CLS 005", "The constructor code for this design exists already."},  
{"CLS 006", "The constructor cannot be defined before the content of the design  
is known."},  
{"CLS 007", "This design has already a constructor."},  
{"CLS 008", "This design does not have a constructor."},  
{"CLS 009", "This design has duplicate messages entries."},  
{"CLS 010", "Unknown design."},  
{"CLS 011", "This design has already an exception list."},  
{"CLS 012", "The constructor for this design has already an exception list."},  
{"CLS 013", "Myself can only be used from within a design."},  
{"CLS 014", "Keep x is only valid here if x is a design parameter."},  
{"CLS 015", "The base class with this name and parameters was not found."},
```

```

{"CLS 016", "There are duplicates in the variable names (ancestor data and
parameters ?)."},
{"CLS 017", "In new x on y, y must be of the type cell."},
{"CLS 018", "attach / detach are only valid for cells. "},
{"CLS 019", "Delete is only valid for cells. "},
{"CLS 020", "'system' can only be used from within a design."},
{"CLS 021", "The destructor code for this design exists already."},
{"CLS 022", "The destructor cannot be defined before the content of the design is
known."},
{"CLS 023", "This design does not have a destructor."},
{"CLS 024", "Destructors do not have an exception table."},
{"CLS 025", "The design is known but with a different signature."},
{"CLS 026", "Inheritance is only possible from a fully defined ancestor."},

```

#### // ERRORS WITH EXTERNAL OBJECTS

```

{"EXT 001", "The library cannot be found (missing load statement ?)."},
{"EXT 002", "The external class method cannot be found in the library."},
{"EXT 003", "No matching parameter pattern found for this external method."},
{"EXT 004", "This library name is already in use for a different DLL."},
{"EXT 005", "This Library was not loaded (not a library ?)."},
{"EXT 006", "Unknown Library (** probably an internal error **)."},
{"EXT 007", "This type is not part of this library."},
{"EXT 008", "External types can be linked to only one CellSpeak type."},
{"EXT 009", "This group is not a library group - library type equivalence is not
possible."},
{"EXT 010", "The external method cannot be found in the library for this
variable."},
{"EXT 011", "This external class is not found in the libraries."},
{"EXT 012", "Type equivalence is only possible for named types and built-in
types."},
{"EXT 013", "Multiple equivalence links."},
{"EXT 014", "Type mismatch."},
{"EXT 015", "Indirection mismatch."},
{"EXT 016", "Could not find the corresponding CellSpeak type for the external
return type."},
{"EXT 017", " NOT USED "},
{"EXT 018", " NOT USED "},
{"EXT 019", "The maximum nr of parameters allowed for an external method is 24"},
{"EXT 020", "A ptr makes only sense if the type corresponds to an external
type."},
{"EXT 021", "Mismatch between external/CellSpeak pointer (unnecessary cast to ptr
?)"},
{"EXT 022", "Alias does not resolve into a library file - maybe check the project
file."},
{"EXT 023", "This group is already linked to a library file."},
{"EXT 024", "Couldn't open the library file."},
{"EXT 025", "Filename does not expand to a path."},
{"EXT 026", "This type exists in the library, but is not defined in CellSpeak."},
{"EXT 027", "Type does not exist in library nor in CellSpeak."},
{"EXT 028", "This function could not be found in the external libs (check name
and/or parameters)."},
{"EXT 029", "Taking the address (&) only has meaning for parameters of external
functions "},

```



```
// ERRORS WITH CONSTANTS
```

```
{"CON 001", "This operation on constants is not supported."},  
{"CON 002", "This unary operation on constants is not supported."},  
{"CON 003", "Only constant declarations are allowed here."},  
{"CON 004", "The value to assign to a constant must be a constant value or an  
expression of constant values. "},  
{"CON 005", "Format of this numerical constant cannot be converted."},  
{"CON 006", "A minus sign for an unsigned value is ignored."},
```

```
//TYPE CAST ERRORS
```

```
{"CST 001", "This type cast is not defined: "},  
{"CST 002", "The default type for this expression list could not be  
determined."},  
{"CST 003", "The type cast to this default type is not defined."},  
{"CST 004", "No conversion possible to a vector or matrix type."},  
{"CST 005", "No conversion possible from the expression list to the record  
type."},  
{"CST 006", "Type casts only need types, no .ptr is required."},  
{"CST 007", "A constant value does not have a runtime address - & value is not  
possible."},  
{"CST 008", "This transient result cannot be cast to a pointer."},  
{"CST 009", "No cast possible between these record types."},  
{"CST 010", "Cast from void to record type - only possible for pointer."},  
{"CST 011", "Cast to a derived record type - allowed, but potentially  
dangerous."},  
{"CST 012", "Cast to class - only possible for null."},
```

```
// CONVERSION ERRORS
```

```
{"CVR 001", "No conversion possible between values, types are not compatible."},  
{"CVR 002", "Conversion failed."},  
{"CVR 003", "Value conversion not defined."},  
{"CVR 004", "No conversion possible for component in expression list to element  
of array."},  
{"CVR 005", "Conversion failed for an element in the constant array."},  
{"CVR 006", "No conversion possible from the component type in the initialiser  
list to the field type in the record."},  
{"CVR 007", "Not enough components for record initialisation."},  
{"CVR 008", "Too many components for record initialisation."},  
{"CVR 009", "Component type cannot be used in conversion to matrix or  
matrix.double."},  
{"CVR 010", "This component type cannot be used as a part of a vector or matrix  
type."},  
{"CVR 011", "Not enough components for this type."},  
{"CVR 012", "Too many components for this type."},  
{"CVR 013", "Conversion of the constant value type to the target type is not  
defined."},  
{"CVR 014", "Conversion of the scalar type to the target type is not defined."},  
{"CVR 015", "Conversion of arguments in this operation is not defined."},  
{"CVR 016", "Unsuccessful value conversion in an assignment to a constant."},
```



```
{"CVR 017", "The record type of the array elements and the record type in the expression list are not the same."}, {"CVR 018", "The component type in the expression list cannot be converted to the component type of the target type."}, {"CVR 019", "There are not enough components in the expression list for the conversion to the target type."}, {"CVR 020", "There are too many components in the expression list for the conversion to the target type."}, {"CVR 021", "Conversion between two string or character encodings was not successful"}, {"CVR 022", "This component type cannot be converted to the field type."}, {"CVR 023", "No conversion exists between these types."},
```

#### // FUNCTION ERRORS

```
{"FCT 001", "Code for this function is already available."}, {"FCT 002", "The names for the parameters do not match with a previous declaration of this function."}, {"FCT 003", "Parameter name re-use is not allowed."}, {"FCT 004", "Function was not found in the symbol table."}, {"FCT 005", "The value cannot be cast to the function return type."}, {"FCT 006", "A return statement is only allowed in a function, a constructor or a destructor."}, {"FCT 007", "The function should return a result."}, {"FCT 008", "This function is declared internal but it is not an internal function."}, {"FCT 009", "This variable is not of the function type and cannot be called."}, {"FCT 010", "The number of parameters for this function does not match."}, {"FCT 011", "Parameter type mismatch."}, {"FCT 012", "The expected return type is a reference type."}, {"FCT 013", "The string constant does not resolve to a function name."}, {"FCT 014", "This function has already an exception list."}, {"FCT 015", "The return type does not match the return type of the declaration."}, {"FCT 016", "A function can only be like a signature."}, {"FCT 017", "Keep x in y : y must be a pool variable."}, {"FCT 018", "Variable to keep is not unique in function."}, {"FCT 019", "The return type of the internal function does not match."}, {"FCT 020", "A full function definition must be followed by an implementation."}, {"FCT 021", "Keep variable has no effect outside a function or message."}, {"FCT 022", "To keep a local variable, use 'keep' in the declaration"}, {"FCT 023", "Keep (static data) is only possible in a design specification context."}, {"FCT 024", "The variable name is already in use for a type that is not a function."}, {"FCT 025", "The variable name and signature is not unique in the current scope."}, {"FCT 026", "This function does not have a matching declaration."}, {"FCT 027", "The variable should be of type function. "}, {"FCT 028", "Too many return values."}, {"FCT 029", "Not enough return values."}, {"FCT 030", "An 'out' parameters cannot be a value, but must be a variable or part of a variable."},
```

```

{"FCT 031", "The 'out' parameters for this field function are different from the
declaration."},
{"FCT 032", "The declaration of a function and its implementation must be in the
same bytecode module."},
{"FCT 033", "Re-declaration of the function."},
{"FCT 034", "Record methods cannot have private data fields."},
{"FCT 035", "Only fields of type function can have code."},
{"FCT 036", "Parameter mismatch between this variable function call and it's
declaration."},
{"FCT 037", "The parameter must have a name."},
{"FCT 038", "There are several functions of this name - specify the parameter
signature."},
{"FCT 039", "Function signature mismatch - no assignment possible."},

```

#### // ERRORS WITH MESSAGES

```

{"MSG 001", "Re-use of a parameter name in a message definition is not
allowed."},
{"MSG 002", "Message(s) with this name found for this class but not with a
different parameter list(s)."},
{"MSG 003", "There are no messages of this name for this class."},
{"MSG 004", "A yield statement has no effect here."},
{"MSG 005", "A drop statement has no effect here."},
{"MSG 006", "The type to the left of the arrow <- should be a cell or a list of
cells."},
{"MSG 007", " *** not used anymore "},
{"MSG 008", "The type to the right of the arrow <- should be a message or a list
of messages."},
{"MSG 009", "This variable or expresssion is not of the message type. "},
{"MSG 010", "This message has already an exception list."},
{"MSG 011", "A flow statement has no effect here."},
{"MSG 012", "the keyword 'Sender' is only valid in message code."},
{"MSG 013", "A flush statement has no effect here."},
{"MSG 014", "Keep x is only valid here if x is a message parameter."},
{"MSG 015", "Design variable is of a different type then this parameter."},
{"MSG 016", "Message name segments must all be fixed names or string literals."},
{"MSG 017", "An expression as a message name must have a result of the type
string."},

```

#### // ERRORS WITH EXCEPTIONS

```

{"EXC 001", "Re-use of a parameter name in an exception definition is not
allowed."},

```

#### // OPERATION ON TYPES

```

{"OPR 001", "Unary operation on this type is not defined."},
{"OPR 002", "This operation between arguments is not defined."},
{"OPR 003", "The number of elements for this type has no meaning."},
{"OPR 004", "This operation on these types exists already."},
{"OPR 005", "(Re)Defining this operation for these types is not possible."},
{"OPR 006", "For an assignment operator the type of the result must be the same as
the type of the left side."},

```

```
{"OPR 007", "This operation is ambiguous for these types."},
{"OPR 008", "An operator function should have one or two parameters."},
{"OPR 009", "Re-declaration of an operator function."},
```

#### // MEMORY ALLOCATION ERRORS

```
{"MEM 001", "No valid type to allocate."},
{"MEM 002", "Variable must be a pointer."},
{"MEM 003", "Variable must be a link to a record type or an array type."},
{"MEM 004", "The size of the array has to be specified in the allocation."},
{"MEM 005", "Allocation of a constant is not possible."},
{"MEM 006", "'delete' is only valid for pointers to records."},
{"MEM 007", " NOT USED ANYMORE "},
{"MEM 008", "Variable must be of the type record or function (use setsize for arrays)."},
{"MEM 009", "setsize is only valid for arrays."},
{"MEM 010", "A swap is only possible between variables of the same type."},
{"MEM 011", "A swap is only possible between variables that have the same type of allocation - use a,b = b,a instead."},
{"MEM 012", "Cannot cast the allocation to a type that is not derived from the record type of the variable."},
{"MEM 013", "In 'new function' the function must be a function variable, not a constant."},
{"MEM 014", "'out' allocation has no effect here (not in a record method)."},
```

#### // ERRORS WITH RECORDS

```
{"REC 001", "No assignment possible between the record field and the corresponding component in the expression list."},
{"REC 002", "No assignment possible : there are more components in the expression list than fields in the record."},
{"REC 003", "No assignment possible : there are more fields in the record than components in the expression list."},
{"REC 004", "No field assignment possible in record : incompatible field types."},
{"REC 005", "No assignment possible : incompatible record types."},
{"REC 006", "This field is already defined for the record."},
{"REC 007", "This type is not a record type."},
{"REC 008", "This field is not part of the record definition."},
{"REC 009", "This field does not exist for the standard type."},
{"REC 010", "Parameter name for the function cannot be the same as a field name for the record."},
{"REC 011", "The field is not a function field."},
{"REC 012", "The type to inherit from is not a record type."},
{"REC 013", "The field of the record is a record of unknown size."},
{"REC 014", "Only a record reference can be assigned NULL."},
{"REC 015", "This variable has no fields."},
{"REC 016", "No such field or function for this type."},
{"REC 017", "This is not a name of a field of this record."},
{"REC 018", "Reference to 'this' only allowed from inside a record or with list."},
{"REC 019", "No such method for this class."},
```

```
{"REC 020", "The field is a variable function field - to be initialised in variables of the type."}, {"REC 021", "Only record types can have var function fields."}, {"REC 022", "This type cannot have data fields."}, {"REC 023", "Both fields should be of the pointer type."}, {"REC 024", "There is no left hand side to initialize (constant or variable)."}, {"REC 025", "The variable or constant in this initialisation cannot have an explicit type."}, {"REC 026", "Field name clashes with the name of a method for the record."}, {"REC 027", "Only record pointers can be compared."}, {"REC 028", "Only record pointers can be compared with null."},
```

#### // REFERENCE ERRORS

```
{"REF 001", "Reference is only allowed for variables."}, {"REF 002", "The variable in the for loop is already a reference variable."},
```

#### // STATEMENT ERRORS

```
{"STM 001", "Case statement for this type does not exist."}, {"STM 002", "There are duplicate values in the case list."}, {"STM 003", "This type for a case-list is not supported."}, {"STM 004", "Control variable in a for loop must be of type int, word, byte or enumerated."}, {"STM 005", "End value in a for loop must be of type int or enumerated."}, {"STM 006", "Not enough index variables for the dimension of the array."}, {"STM 007", "Too many index variables for the array."}, {"STM 008", "The type of the indexed variable must be an array."}, {"STM 009", "The index variable(s) in a for loop must be of type int or word."}, {"STM 010", "In a 'for each i in enumeration type' loop, there can only be one index."}, {"STM 011", "In a loop: 'for each x in y', y must be of the enumeration type, array type or list type."}, {"STM 012", "Mismatch between a begin/end pair in a compound statement."}, {"STM 013", "The index must be selected with an integer from 1 to the number of indexes of the array."}, {"STM 014", "The selected index is not in the acceptable range for this array."}, {"STM 015", "The loop variable in a for each statement must be of the same type as the array elements."}, {"STM 016", "A parameter cannot be a loop variable."}, {"STM 017", "This type cannot be implicitly compared with 0 or null."}, {"STM 018", "This task label name is already in use."}, {"STM 019", "This name cannot be used as a task name."}, {"STM 020", "This 'leave' statement has no effect."}, {"STM 021", "This 'continue' statement has no effect."},
```

#### // FILE RELATED

```
{"FIL 001", "The message file could not be opened. The internal messages will be used."},
```

```
// TEMPLATE ERRORS
```

```
{"TPL 001", "A template of this name exists already in the current scope"},  
{"TPL 002", "The template could not be parsed successfully."},  
{"TPL 003", "The item to insert is not a template name."},  
{"TPL 004", "This template did not load successfully (check syntax)."},
```

```
// BYTECODE ASSEMBLY ERRORS
```

```
{"BYC 001", "Unknown bytecode instruction."},  
{"BYC 002", "Unknown special bytecode variable."},  
{"BYC 003", "Operand mismatch for this bytecode instruction."},  
{"BYC 004", "Too many operands for this instruction."},  
{"BYC 005", "Operand cannot be a constant."},  
{"BYC 006", "Operand not a constant or of the wrong type."},  
{"BYC 007", "Operand not supported."},  
{"BYC 008", "Not enough operands for this instruction."},  
{"BYC 009", "Offset must be an integer constant."},
```